Joo-Young Kim
Bongjin Kim
Tony Tae-Hyoung Kim *Editors*

Processing-in-Memory for AI

From Circuits to Systems

Springer

Processing-in-Memory for AI

Joo-Young Kim • Bongjin Kim
Tony Tae-Hyoung Kim
Editors

Processing-in-Memory for AI

From Circuits to Systems

Springer

Editors
Joo-Young Kim
Korea Advanced Institute of Science and Technology
Daejeon, Korea (Republic of)

Bongjin Kim
University of California
Santa Barbara, CA, USA

Tony Tae-Hyoung Kim
Nanyang Technological University
Nanyang, Singapore

ISBN 978-3-030-98780-0 ISBN 978-3-030-98781-7 (eBook)
https://doi.org/10.1007/978-3-030-98781-7

Contents

Chapter 1
Introduction

Joo-Young Kim

1.1 Hardware Acceleration for Artificial Intelligence and Machine Learning

Artificial intelligence (AI) and machine learning (ML) technology enable computers to mimic the cognitive tasks believed to be what only humans can do, such as recognition, understanding, and reasoning [1]. A deep ML model named AlexNet [2], which uses eight layers in total, won the famous large-scale image recognition competition called ImageNet by a significant margin over shallow ML models in 2012. Since then, deep learning (DL) revolution has been ignited and spread to many other domains such as speech recognition [3], natural language processing [4], virtual assistance [5], autonomous vehicle [6], and robotics [7]. With significant successes in various domains, DL revolutionizes a wide range of industry sectors such as information technology, mobile communication, automotive, and manufacturing [8]. However, as more industries adopt the new technology and more people use it daily, we face an ever-increasing demand for a new type of hardware for the workloads. Conventional hardware platforms such as CPU and GPU are not suitable for the new workloads. CPUs cannot cope with the tremendous amount of data transfers and computations required in the ML workloads, while GPUs consume large amounts of power with high operating costs.

AI chip or accelerator is the hardware that enables faster and more energy-efficient processing for AI workloads (Fig. 1.1). Over the past few years, many AI accelerators have been developed to serve the new workloads, targeting from battery-powered edge devices [9–11] to datacenter servers [12]. As McKinsey predicted in the report [13], the AI semiconductor industry is expected to grow 18–19%

J.-Y. Kim (✉)
School of Electrical Engineering (E3-2), KAIST, Daejeon, South Korea
e-mail: jooyoung1203@kaist.ac.kr

J.-Y. Kim et al. (eds.), *Processing-in-Memory for AI*,
https://doi.org/10.1007/978-3-030-98781-7_1

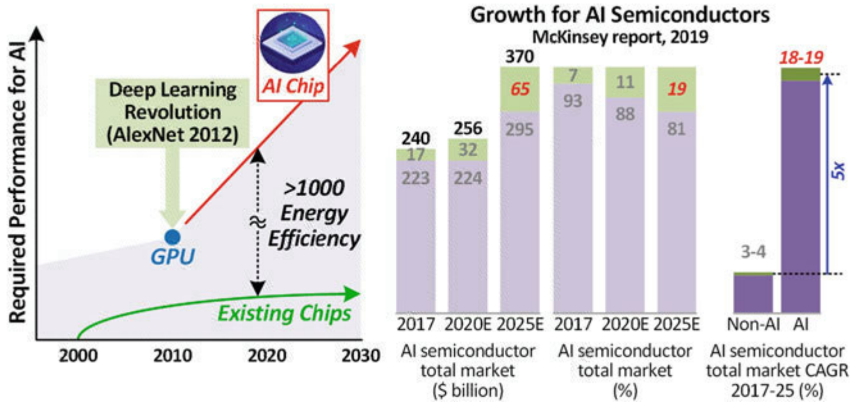

Fig. 1.1 AI chip and its market prediction

every year to 65 billion, accounting for about 19% in the entire semiconductor market in 2025. So far, the AI hardware industry is led by big tech companies. Google developed their own AI chip named tensor processing unit (TPU) that can work with TensorFlow [14] software framework. Amazon developed Inferentia chip [15] for high-performance ML inference. Microsoft's BrainWave [16] uses FPGA infrastructure to accelerate ML workloads at scale. Even an electric car maker Tesla developed the full self-driving (FSD) chip for autonomous vehicles. There are many start-up companies in this domain. Habana Labs, acquired by Intel in late 2019, developed Gaudi processor for AI training. Graphcore has developed intelligent processing unit (IPU) [17] and deployed in datacenters. Groq's tensor streaming processor [18] optimizes data streaming and computations with fixed task scheduling. Cerabras's wafer-scale engine [19] tries to use a whole wafer as a ML processor to keep a large model without external memories.

1.2 Machine Learning Computations

In this section, we introduce the basic models of deep neural networks (DNNs) and their computations. A DNN model is composed of multiple layers of artificial neurons, where neurons of each layer are inter-connected with the neurons in the neighbor layers. The mathematical model of the neuron comes from Frank Rosenblatt's Perceptron [20] model, as shown in Fig. 1.2. Inspired by the human neuron model, it receives multiple inputs among many input neurons and accumulates their weighted sums with a bias. Then it decides the output through an activation function, where the activation function is non-linear and differentiable, having a step-like characteristic shape. As a result, the output of a neuron is expressed with the following equation:

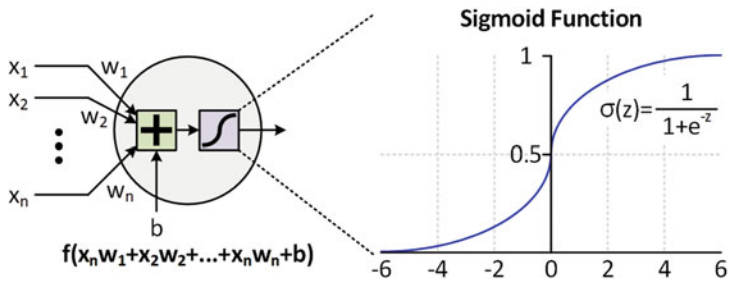

Fig. 1.2 Artificial neuron: perceptron model

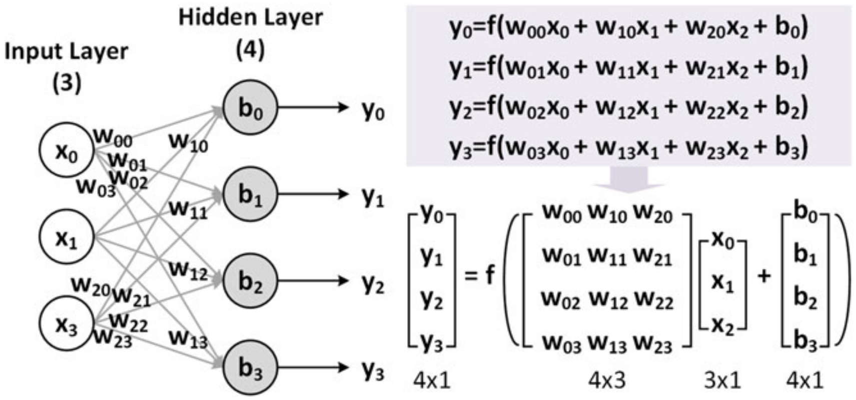
Input Layer (3)
Hidden Layer (4)
x0
x1
x3
b0
b1
b2
b3
y0
y1
y2
y3
w00 w01 w02 w03 w10 w11 w12 w13 w20 w21 w22 w23
y0=f(w00x0 + w10x1 + w20x2 + b0)
y1=f(w01x0 + w11x1 + w21x2 + b1)
y2=f(w02x0 + w12x1 + w22x2 + b2)
y3=f(w03x0 + w13x1 + w23x2 + b3)
4x1
4x3
3x1
4x1

Fig. 1.3 Fully connected layer

$$y = f(w_1x_1 + w_2x_2 + \ldots + w_nx_n + b) \tag{1.1}$$

Based on the network connection, there are three major layers in the DNN models, which determines the actual computations: fully connected, convolutional, and recurrent layer.

1.2.1 Fully Connected Layer

Figure 1.3 shows the fully connected layer that interconnects the neurons in the input layer to the neurons in the next layer. The input vector is the values of input neurons, a 3×1 vector in this case, and the output vector is the values of output neurons, a 4×1 vector. Each connection of the fully connected network between the two layers represents a weight parameter in the model. For example, W_{01} represents a weight parameter of the connection between the input neuron 0 and the output neuron 1. Collectively, the network becomes a 4×3 weight matrix. In addition, each output neuron has a bias, so the layer has 4×1 bias vector. For each output neuron, we can

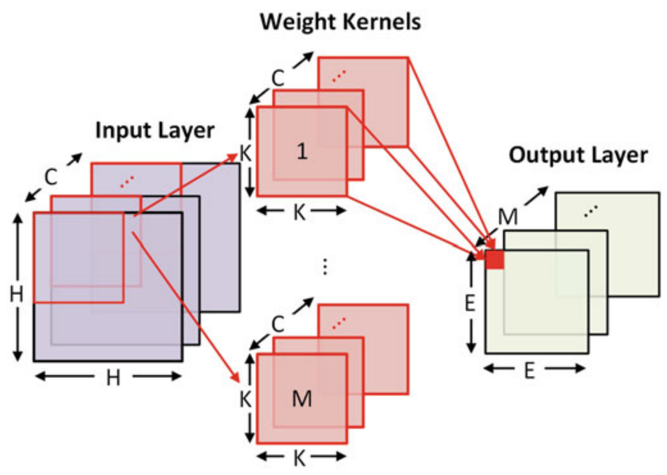

Fig. 1.4 Convolutional layer

write the output value y using Eq. 1.1. As a result, the equations can be formulated into a matrix-vector equation as follows:

$$y = f(Wx + b) \tag{1.2}$$

If the model includes multiple layers, which is the case of deep neural networks, the matrix operations will be cascaded one by one. Traditional multi-layer perceptron (MLP) models as well as the latest transformer models [21] are based on the fully connected layer.

1.2.2 Convolutional Layer

The convolutional layer iteratively performs 3-d convolution operations on the input layer using multiple weight kernels to generate the output layer, as illustrated in Fig. 1.4. The input layer has multiple 2-d input feature maps, sized $H \times H \times C$, and the size of each kernel is $K \times K \times C$. For computation, it performs 3-d convolution operations from top-left to bottom-right for each kernel with a stride of U. A single convolution operation accumulates all the inner products between the input and the kernel. As a result of scanning for a kernel, it gets a single output feature map sized $E \times E$. By repeating this process for all kernels, the convolutional layer produces the final output layer, sized to $E \times E \times M$. The equation for an output point in the convolutional layer is as follows:

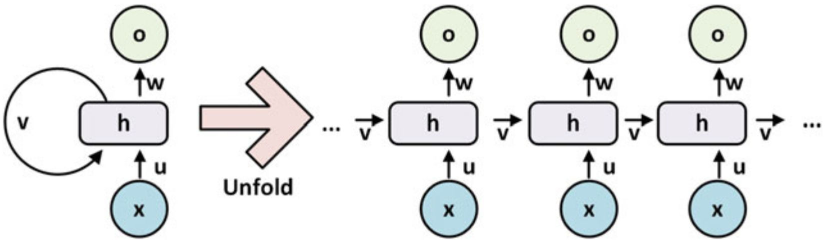

Fig. 1.5 Recurrent layer

$$\mathbf{O}[u][x][y] = \mathbf{B}[u] + \sum_{k=1}^{C-1}\sum_{i=1}^{K-1}\sum_{j=1}^{K-1} \mathbf{I}[k][Ux+i][Uy+j]\mathbf{W}[u][k][i][j],$$
$$0 \le u < M, 0 \le x, y < E, E = \frac{(H - R + U)}{U} \tag{1.3}$$

Many convolution neural network (CNN) models use a number of convolutional layers with a few fully connected layers at the end for image classification and recognition task [22].

1.2.3 Recurrent Layer

Figure 1.5 shows the recurrent layer that has a feedback loop from the output to the input layer in the fully connected setting. In this layer, the cell state of the previous timestamp affects the current state. Its computation is also matrix-vector multiplication but involves multiple steps with dependency. Its cell and output value are expressed as follows. The hyperbolic tangent is usually used for activation function in the recurrent layer.

$$h_t = f(U_h x_t + V_h h_{t-1} + b_h), O_t = f(W_h h_t + b_o) \tag{1.4}$$

Recurrent neural networks (RNNs) such as GRUs [23] and LSTMs [24] are based on this type of layer and popularly used for speech recognition.

1.3 von Neumann Bottleneck

1.3.1 Memory Wall Problem

Von Neumann architecture [25] is a computer architecture proposed by John von Neumann in 1945, which broadly consists of a compute unit that executes a program written by a user and a memory unit that stores both the user's program and data required to run the program. Most modern computer systems, including CPU and GPU, fall into this architecture. With the Moore's law that states the number of transistors on a computer chip doubles every 18 months [26] and the process technology scaling, its compute performance has been rapidly improved, as shown in Fig. 1.6. On the other hand, the memory device has been developed to increase its capacity, not the performance. Therefore, the performance gap between the two separated devices gets wider and wider, and it becomes a major performance issue in the system. This memory wall problem causes the data movement issue or limits the memory bandwidth between the compute and memory device. It is often called von Neumann bottleneck because all the computers with von Neumann architecture inevitably have this bottleneck simply because they have separated compute and memory devices.

1.3.2 Latest AI Accelerators with High-Bandwidth Memories

The von Neumann bottleneck has been mitigated with a hierarchical memory structure. Processors include the fastest but smallest SRAM-based cache on-chip to leverage the temporal and spatial locality. Outside of the processor chip, there exists the main memory of the system based on DRAM. DRAM is fast and has a larger capacity than SRAM. After that, the system has solid-state drives (SSD) for high storage capacity. However, as the DNN models get deeper and bigger to the tera-bytes level, the ML workloads require even higher bandwidth between the processor chip and the main memory. Even worse, the process technology scaling

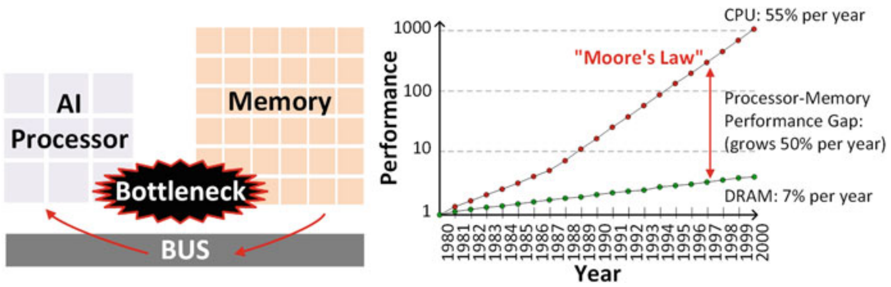

Fig. 1.6 von Neumann Bottleneck and memory wall problem

faces strong challenges with the end of Moore's law below 10 nm technology node [27].

To overcome the von Neumann bottleneck and the slow-down of process scaling, many companies propose array-type architectures to accelerate data-intensive ML processing along with 3-d stacking DRAM technology called high-bandwidth memory (HBM) [28] to provide higher bandwidth between the compute and memory devices. Google developed their own AI chip named TPU to serve the inference and training workloads in datacenters with a better cost and energy efficiency [12]. Intel recently released the NNP-T processor [29] and Habana Labs Gaudi processor [30] for training workloads. Start-up companies such as Graphcore [31] and Groq [18] are also based on this architecture. However, although these AI accelerators with HBM technology can mitigate the bandwidth bottleneck up to a couple TB/s level, they cannot address it eventually as they still fall into von Neumann architecture. In addition, the HBM suffers from high-power dissipation and low capacity [32]. Table 1.1 shows the summary of the hardware specifications of the latest AI accelerators [33].

1.4 Processing-in-Memory Architecture

1.4.1 Paradigm Shift from Compute to Memory

An architectural paradigm called processing-in-memory (PIM) takes an alternative approach to the conventional von Neumann architecture to solve the memory bandwidth problem. It is not a new concept, as it is first introduced in 1970s [34] and has many subsequent works [35, 36]. Recently, PIM has gotten increasing attention amid the memory wall crisis caused by modern ML applications requiring high bandwidths.

In PIM architecture, instead of fetching data from the memory unit to compute unit, data stays in the memory, while the merged logic performs computations in place without moving data outside. As Fig. 1.7 illustrates, this approach is a radical change in the computer architecture; the traditional and near-memory architectures basically have the same memory hierarchy to utilize the external memory bandwidth efficiently, but the PIM merges the compute and memory devices so that it does not have any problems in external data movement. This fundamental shift from compute-centric architecture to memory-centric or combined architecture gets much attention to solve the von Neumann bottleneck, especially for data-intensive applications such as AI and ML. On top of performance improvement, it saves significant energy by replacing expensive external data transfers with on-chip data movements.

Table 1.1 Latest AI accelerators

Metric	Google TPU v3	Nvidia V100	Nvidia A100	Cerebras WSE	GraphCore IPU1	GraphCore IPU2
Technology node	12 nm (16 nm est.)	TSMC 12 nm	TSMC 7 nm	TSMC 16 nm	TSMC 16 nm	TSMC 7 nm
Die area (mm^2)	648 (600 est.)	815	826	46225	900 (est.)	823
Transistor count (B)	11 (est.)	21	54.2	1200	23.6	59.4
Architecture	Systolic array	SIMD + TC	SIMD + TC	MIMD	MIMD	MIMD
Theoretical TFLOPS (16-bit mixed precision)	123	125	312	2500	125	250
Freq (GHz)	0.92	1.5	1.4	Unknown	1.6	Unknown
DRAM capacity (GB)	32	32	80	N/A	N/A	112
DRAM BW (GB/s)	900	900	2039	N/A	N/A	64 (est.)
Total SRAM capacity	32 MB	36 MB (RF+L1+L2)	87 MB (RF+L1+L2)	18 GB	300 MB	900 MB
SRAM BW (TB/s)	Unknown	224 @RF +14 @L1 +3 @L2	608 @RF +19 @L1 +7 @L2	9000	45	47.5
Max TDP (W)	450	450	400	20K	150	150 (est.)

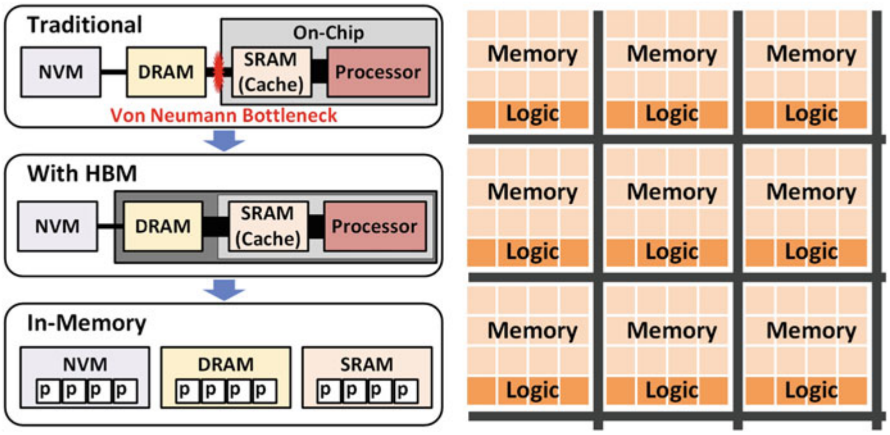

Fig. 1.7 Processing-in-memory architecture

1.4.2 Challenges

Although it looks promising, PIM has many challenges as it needs to integrate logic units into the memory module. The three notable challenges in PIM design are process accessibility, architecting, and designing considering physical constraints, and software stack and usability.

Among many memory technologies, SRAM is the only memory type that we can build using a commercially available logic process. This is why many PIM prototypes are based on SRAM [37–40]. It is possible to fabricate with a logic process and easy to customize both memory cell and peripheral circuits. As the cell size is the biggest among others, SRAM-based PIM has the least area restriction on the logic integration. Except SRAM, DRAM and non-volatile memory (NVM) processes are difficult to access. Memory vendors such as Samsung, SK Hynix, and Micron have their own memory processes, but they are not open to outside. Since the process design kit (PDK) is not accessible, most researchers cannot even simulate the basic circuits. There have been many PIM architecture proposals for DRAM [41, 42]; however, they only evaluate the architectures at a performance simulator level without much physical design. Since the DRAM process is vastly different from the logic process, focusing on increasing cell capacity and cell density, it is hard to convince that the proposed PIM architectures are feasible to be fabricated with only simulations.

It is imperative for chip designers to choose what function they should put into the memory in the PIM design. They cannot implement various functions or too generic logic as the silicon area is limited. In addition, the chip will lose the memory capacity for the area of the logic merged. Another challenge is that the logic design should be physically aligned with the memory cell design to maximize the internal bandwidth.

Table 1.2 PIM opportunities and challenges

PIM opportunities	PIM challenges
1. Non-von Neuman Architecture → Can solve von Neumann bottleneck. 2. Converged Logic + Memory → Can achieve high internal bandwidth. 3. Suitable for data-intensive workloads → Good for AI/ML applications 4. Little external data movement → Can achieve high energy efficiency	1. Process accessibility : Memory process is difficult to use 2. Limited area resource : What function logic should the designer add? 3. Physical layout constraint : To maximize the internal compute bandwidth 4. SW stack for PIM deployment : Revisit a whole SW stack for wide adoption

The software stack is the last hurdle in the PIM design. It is essential for the wide-spreading adoption of PIM as a new device. Unlike traditional memory devices, PIM is not a passive device anymore as it can perform logic operations at the same time. What this means is that we need a fundamental change in the software side either. For real PIM system optimizations, we need to revisit a whole software stack, including programming language, compiler, driver, and run-time. Otherwise, it will not be able to outperform the existing von Neumann computer's performance and usability. Table 1.2 summarizes the opportunities and challenges of PIM technology.

1.5 Book Organization

This book organizes as follows. In Chap. 2, we study the backgrounds of the PIM technology, including basic memory operations of various memories such as SRAM, DRAM, and Resistive RAM (ReRAM). We also discuss the PIM's design constructions and approaches in this chapter. From Chaps. 3–5, we will investigate significant PIM designs in the major memory technologies: SRAM, DRAM, and ReRAM. Each chapter will cover comprehensive design technologies required for PIM, including in-memory circuit processing, memory macro design, data mapping strategy, and architecture. In Chap. 6, we will focus on the PIMs designed for ML training. We will discuss the systems side of PIM, including software and programming interface, in Chap. 7, for the wide adoption of the technology. Finally, we will conclude our book with future remarks in Chap. 8.

References

1. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. Nature **521**(7553), 436–444 (2015)
2. A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks. Adv. Neural Inform. Process. Syst. **25**, 1097–1105 (2012)

3. G. Hinton, L. Deng, D. Yu, G.E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
4. Y. Goldberg, Neural network methods for natural language processing. Synth. Lect. Hum. Lang. Technol. **10**(1), 1–309 (2017)
5. V. Kepuska, G. Bohouta, Next-generation of virtual personal assistants (Microsoft Cortana, Apple Siri, Amazon Alexa and Google Home), In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, Piscataway (2018), pp. 99–103
6. M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, R. Urtasun, MultiNet: real-time joint semantic reasoning for autonomous driving, in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, Piscataway (2018), pp. 1013–1020
7. H.A. Pierson, M.S. Gashler, Deep learning in robotics: a review of recent research. Adv. Robot. **31**(16), 821–835 (2017)
8. M.I. Jordan, T.M. Mitchell, Machine learning: trends, perspectives, and prospects. Science **349**(6245), 255–260 (2015)
9. Y.H. Chen, T. Krishna, J.S. Emer, V. Sze, Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE J. Solid-State Circuits **52**(1), 127–138 (2016)
10. Z. Yuan, Y. Liu, J. Yue, Y. Yang, J. Wang, X. Feng, J. Zhao, X. Li, H. Yang, STICKER: an energy-efficient multi-sparsity compatible accelerator for convolutional neural networks in 65-nm CMOS. IEEE J. Solid-State Circuits **55**(2), 465–477 (2019)
11. J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, H.J. Yoo, UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision, in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, Piscataway (2018), pp. 218–220
12. N.P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T.V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C.R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, D.H. Yoon, In-datacenter performance analysis of a tensor processing unit, in *Proceedings of the 44th Annual International Symposium on Computer Architecture* (2017), pp. 1–12
13. G. Batra, Z. Jacobson, S. Madhav, A. Queirolo, N. Santhanam, *Artificial-Intelligence Hardware: New Opportunities for Semiconductor Companies*. McKinsey and Company (2019)
14. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: a system for large-scale machine learning, in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (2016), pp. 265–283
15. Mitchell TM, Machine learning, in Amazon (2017). https://aws.amazon.com/machine-learning/inferentia/. Accessed 21 Oct 2021
16. E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, M. Abeydeera, L. Adams, H. Angepat, C. Boehn, D. Chiou, O. Firestein, A. Forin, K.S. Gatlin, M. Ghandi, S. Heil, K. Holohan, A. El Husseini, T. Juhasz, K. Kagi, R.K. Kovvuri, S. Lanka, F. van Megen, D. Mukhortov, P. Patel, B. Perez, A.G. Rapsang, S.K. Reinhardt, B.D. Rouhani, A. Sapek, R. Seera, S. Shekar, B. Sridharan, G. Weisz, L. Woods, P.Y. Xiao, D. Zhang, R. Zhao, D. Burger, Serving DNNs in real time at datacenter scale with project brainwave. iEEE Micro **38**(2), 8–20 (2018)
17. Ltd G IPU processors, in: *IPU Processors*. https://www.graphcore.ai/products/ipu. Accessed 21 Oct 2021